Computación

Subrutinas

- Permiten manejar cálculos repetitivos.
- Se usan cuando no tengo una estructura de función.
- Devuelven más de un resultado.
- Ahorran mucho tiempo al programar.

¿Cómo se usan?

```
Program Prueba
REAL*4 C(100)
Integer B
⋮
CALL TEST(A, B, C)
⋮
END

SUBROUTINE TEST(X,Y,Z)
REAL*4 Z(100)
Integer Y
⋮
RETURN
END
```

Ventajas, propiedades y formas de uso:

- Se escriben en forma externa al programa, como lo hacen las funciones externas.
- Intercambian con el programa principal una lista de variables. A esas variables se las llama argumentos.
- Los argumentos son variables que se envían del programa a la subrutina y cuando esta termina son devueltos. Sólo esa lista es intercambiada. No se define cuales de estos argumentos son datos que ingresan información o cuales variables son las que devuelven los resultados.
- —> Los argumentos son una lista de variables que se intercambia con la subrutina, donde cualquiera de estas variables puede ser modificada o no.

Ventajas, propiedades y formas de uso:

- Las subrutinas son llamadas del programa principal a partir de la orden **CALL Nombre de la subroutina(lista de variables)**.

- No son parte de un cálculo o una asignación en una sentencia, como si lo hacen las funciones.

- Las subrutinas se escriben al final del programa principal en el mismo archivo o en archivos separados que se compilan junto al principal. Se debe indicar como primer sentencia la orden: **SUBROUTINE Nombre de la subrutina(lista de variables)**.

- —> Se escriben en forma externa al programa, como lo hacen las funciones externas o en un archivo separado que debo compilar con el programa.

Ventajas, propiedades y formas de uso:

- Los argumentos se copian a variables en el espacio de memoria asignado a la subrutina y cuando esta termina se copian de nuevo a la memoria del programa. Las variables internas de la subrutina desaparecen en el momento que esta termina.

- Al igual que en las funciones externas, es necesario que las variables en los argumentos de llamada de la subrutina sean del mismo tipo y dimensión en el programa y en la subrutina. Por eso, el orden de cada variable en los argumentos debe ser el mismo en el programa principal y en la subrutina para mantener la coherencia.

- La subrutina retorna al programa cuando ejecuta la sentencia **RETURN**. Este retorno se hace exactamente al lugar donde fue llamada. Un programa puede llamar a una subrutina todas las veces que sea necesario.

Ventajas, propiedades y formas de uso:

- La última sentencia de una subrutina es al igual que cualquier otro programa, es la orden **END**.
- Una subrutina puede llamar a otra subrutina y cuando termina devuelve el control a la que la llamó en el punto que la llamó.
- Una subrutina puede llamarse a sí misma. Esta propiedad se llama **recursividad**.
- Si en el programa existe una llamada a una subrutina en particular pero esta no es encontrada por el compilador se señalará como error. No se indica como error la existencia de una subrutina que no es usada por el programa principal u otras subrutinas.

Ventajas, propiedades y formas de uso:

- Las subrutinas pueden no estar escritas en el mismo lenguaje que el programa que las llama, pero sí se deberán respetar el tipo de variable, su dimensión y el orden de los argumentos en la llamada.

- Las subrutinas pueden precompilarse en grupos y estos archivos se los denomina Librerías. Los sistemas operativos tienen muchas librerías que son accesibles desde los programas, incluso algunas de estas, las consideradas esenciales están cargadas en la memoria ram.

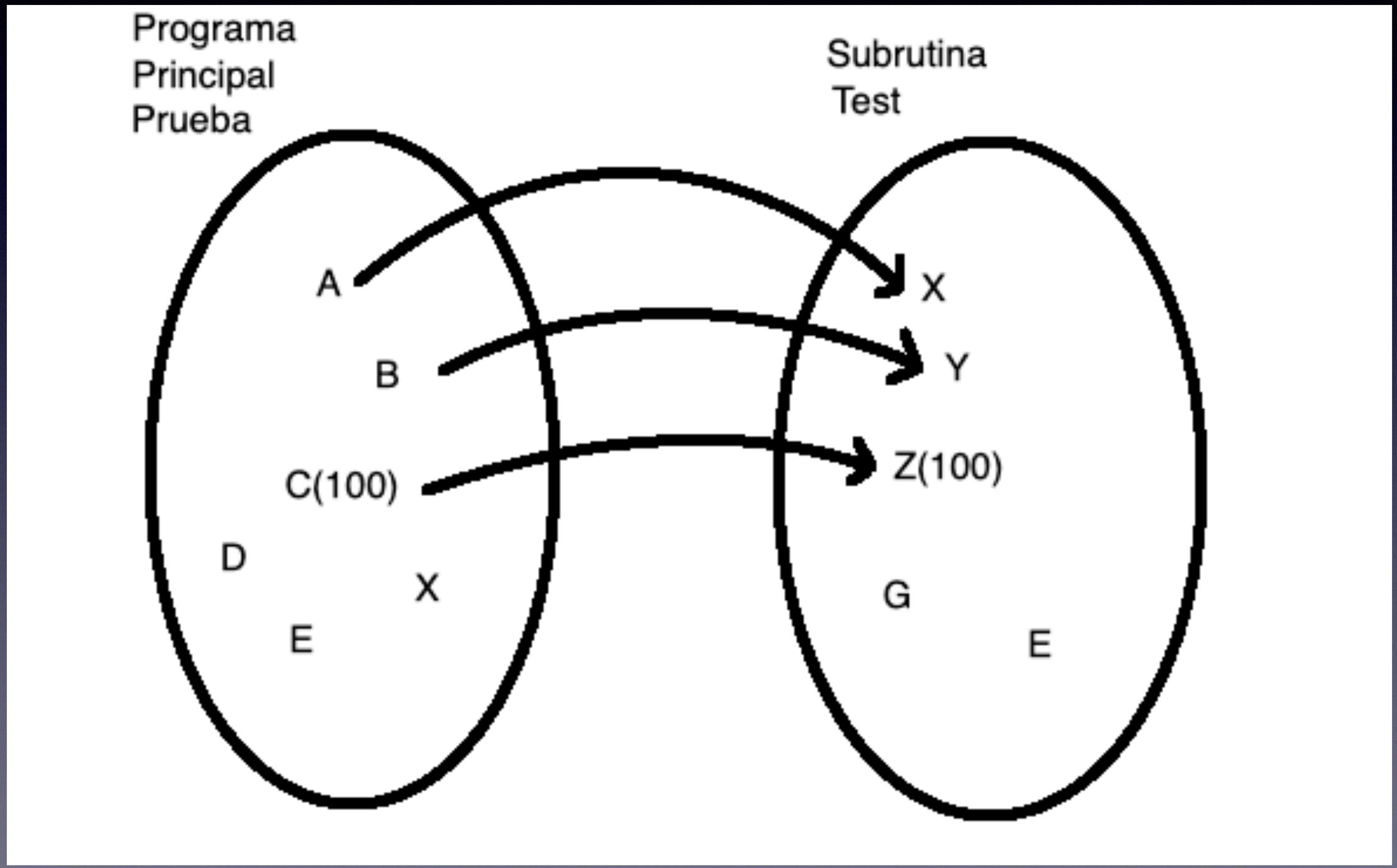
- Por ejemplo, existen librerías para que los programas Fortran puedan hacer dibujos (PGPLOT, etc) o hacer cálculos vectoriales (BLAS, etc).

¿Cómo se usan?

```
Program Prueba
REAL*4 C(100)
Integer B
⋮
CALL TEST(A, B, C)
⋮
END

SUBROUTINE TEST(X,Y,Z)
REAL*4 Z(100)
Integer Y
⋮
RETURN
END
```

¿Cómo se utilizan los argumentos desde el punto de vista de la memoria?



C. Feinstein

Coordenadas polares

```
program polares
pi=atan(1)*4

write(*,*)'Ingrese X,Y'
read(*,*) X,Y

call pola(X,Y,r,theta)
write(*,*)'R =',r,'Theta =', theta/pi*180

call cartesianas(X,Y,r,theta)
write(*,*)'x =',X,'y =',Y

end

Subroutine pola(x,y,r,theta)
r = sqrt(x*x + y*y)
theta = atan2(y,x)
return
end

Subroutine cartesianas(x,y,r,theta)
x=r*cos(theta)
y=r*sin(theta)
return
end
```

Programando con subrutinas

- Permiten dividir un programa grande en muchos pequeños más enfocados en un punto en particular
- La mayoría de las subrutinas de algoritmos matemáticos de uso común ya están hechas. Ver por ejemplo el libro Numerical Recipes

Ventajas de las subrutinas ya desarrolladas

- Utilizan menos operaciones matemáticas
- Se probaron muchos algoritmos (no necesariamente los que pensamos que son los mejores)
- Se sabe la cantidad de operaciones que se realizarán con cierta precisión.
- Se predice la calidad numérica del resultado y suele estar documentada

Ventajas del uso de subrutinas

- Permiten trabajar en colaboración de una manera eficiente.
- Se recupera fácilmente código de otros proyectos que son útiles para el actual.
- Permiten reemplazar algoritmos con otros mejores de forma eficiente.
- Se reduce el programa principal a la llamada de las subrutinas en orden o a la parte innovadora que se está trabajando.

Sentencia Common - Include

Permite enviar argumento que no necesariamente esté en el nombre de la subrutina. Sirve cuando se trabaja con muchas variables

common /nombre1/ Lista de variables 1
common /nombre2/ Lista de variables 2

Ejemplo:

```
COMMON /listado1/ A,B,C,IK,X(1000)
COMMON /listado2/ B1,B2,B3,B4

Y en la subroutine pondría
SUBROUTINE SUB1()
COMMON /listado1/ x,y,j,es(1000)
…
…
RETURN
END
```

Recursión

```
program factor
write(*,*)'Cual es el numero:'
read(*,*) n
call factorial(n,p)
write(*,*) p
end

Recursive Subroutine factorial(n,p)
if(n.gt.1) then
Call factorial(n-1,p)
     p=p*n
else
     p=1
endif
return
end
```

Una subrutina puede llamarse a si misma
Eso es recursión

N! = N*(N-1)!

Recursión

```
program factor
write(*,*)'Cual es el numero:'
read(*,*) n
call factorial(n,p)
write(*,*) p
end

Recursive Subroutine factorial(n,p)
if(n.gt.1) then
Call factorial(n-1,p)
     p=p*n
else
     p=1
endif
return
end
```

C. Feinstein

Sentencia Save

Se vuelve a la forma antigua del la forma de manejar la memoria. Las variables se guardan entre llamados a la subrutina.
No se permite la recursión si se usa esta sentencia.

Forma de uso:

Save listado de variables

Save x,y,z